1 Putting ASP.NET Core in context

This chapter covers

- Putting ASP.NET Core in context
- Understanding the role of the ASP.NET Core platform
- Putting the ASP.NET Core application frame works in context
- Understanding the structure of this book
- Getting support when something doesn't work

ASP.NET Core is Microsoft's web development platform. The original ASP.NET was introduced in 2002, and it has been through several reinventions and reincarnations to become ASP.NET Core 7, which is the topic of this book.

ASP.NET Core consists of a platform for processing HTTP requests, a series of principal frameworks for creating applications, and secondary utility frameworks that provide supporting features, as illustrated by figure 1.1.

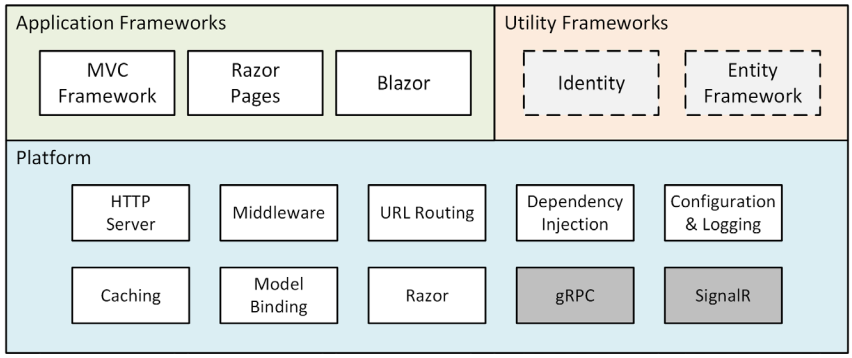

Figure 1.1 The structure of ASP.NET Core

Understanding .NET Core, .NET Framework, and .NET

If you have never worked for a large corporation, you might have the impression that Microsoft is a disciplined organization with a clear strategy and an army of programmers working together to deliver complex products like ASP.NET Core.

In reality, Microsoft is a chaotic collection of dysfunctional tribes that are constantly trying to undermine each other to get prestige and promotions. Products are released during lulls in the fighting, and successes are often entirely unexpected. This isn't unique to Microsoft—it is true of any large company—but it has a particular bearing on ASP.NET Core and the naming confusion that Microsoft has created.

Several years ago, the part of Microsoft responsible for ASP.NET created its own version of the .NET platform, allowing ASP.NET to be updated more often than the rest of .NET. *ASP.NET Core* and *.NET Core* were created, allowing cross-platform development, and using a subset of the original .NET APIs, many of which were specific to Windows. It was a painful transition, but it meant that web development could evolve independently of the "legacy" Windows-only development, which would continue under the renamed *.NET Framework*.

But no one wants to be in the "legacy" tribe because there is no glory in keeping the lights on at Microsoft. .NET Core was clearly the future and, one by one, the.NET groups at Microsoft argued that their technology and APIs should be part of .NET Core. The .NET Core APIs were gradually expanded, and the result was an incoherent mess, with half-hearted attempts to differentiate .NET Core and .NET Framework and standardize the APIs.

To clean up the mess, Microsoft has merged *.NET Core* and *.NET Framework* into *.NET*, dropping the *Core* part of the name. ".NET" is a name I like to think was chosen on the way out of the office on a holiday weekend but which I suspect is the result of many months of heated argument.

The problem with dropping *Core* from the name is that it cannot be carried out consistently. The name *ASP.NET Core* originally denoted the .NET Core version of ASP.NET, and going back to that name would be even more confusing.

(continued)

The result is that even Microsoft can't decide what name to use. You will see the term *ASP.NET Core* in a lot of the developer documentation—and that's the name I use in this book—but you will also see *ASP.NET Core in .NET*, especially in press releases and marketing material. It is not clear which name will win out, but until there is clarity, you should take care to determine whether you are using .NET Framework, .NET Core, or .NET.

1.1 Understanding the application frameworks

When you start using ASP.NET Core, it can be confusing to find that there are different application frameworks available. As you will learn, these frameworks are complementary and solve different problems, or, for some features, solve the same problems in different ways. Understanding the relationship between these frameworks means understanding the changing design patterns that Microsoft has supported, as I explain in the sections that follow.

1.1.1 Understanding the MVC Framework

The MVC Framework was introduced in the early ASP.NET, long before .NET Core and the newer .NET were introduced. The original ASP.NET relied on a development model called Web Forms, which re-created the experience of writing desktop applications but resulted in unwieldy web projects that did not scale well. The MVC Framework was introduced alongside Web Forms with a development model that embraced the character of HTTP and HTML, rather than trying to hide it.

MVC stands for Model-View-Controller, which is a design pattern that describes the shape of an application. The MVC pattern emphasizes *separation of concerns*, where areas of functionality are defined independently, which was an effective antidote to the indistinct architectures that Web Forms led to.

Early versions of the MVC Framework were built on the ASP.NET foundations that were originally designed for Web Forms, which led to some awkward features and workarounds. With the move to .NET Core, ASP.NET became ASP.NET Core, and the MVC Framework was rebuilt on an open, extensible, and cross-platform foundation.

The MVC Framework remains an important part of ASP.NET Core, but the way it is commonly used has changed with the rise of single-page applications (SPAs). In an SPA, the browser makes a single HTTP request and receives an HTML document that delivers a rich client, typically written in a JavaScript framework such as Angular or React. The shift to SPAs means that the clean separation that the MVC Framework was originally intended for is not as important, and the emphasis placed on following the MVC pattern is no longer essential, even though the MVC Framework remains useful (and is used to support SPAs through web services, as described in chapter 19).

Putting patterns in their place

Design patterns provoke strong reactions, as the emails I receive from readers will testify. A substantial proportion of the messages I receive are complaints that I have not applied a pattern correctly.

Patterns are just other people's solutions to the problems they encountered in other projects. If you find yourself facing the same problem, understanding how it has been solved before can be helpful. But that doesn't mean you have to follow the pattern exactly, or at all, as long as you understand the consequences. If a pattern is intended to make projects manageable, for example, and you choose to deviate from that pattern, then you must accept that your project may be more difficult to manage. But a pattern followed slavishly can be worse than no pattern at all, and no pattern is suited to every project.

My advice is to use patterns freely, adapt them as necessary, and ignore zealots who confuse patterns with commandments.

1.1.2 Understanding Razor Pages

One drawback of the MVC Framework is that it can require a lot of preparatory work before an application can start producing content. Despite its structural problems, one advantage of Web Forms was that simple applications could be created in a couple of hours.

Razor Pages takes the development ethos of Web Forms and implements it using the platform features originally developed for the MVC Framework. Code and content are mixed to form self-contained pages; this re-creates the speed of Web Forms development without some of the underlying technical problems (although scaling up complex projects can still be an issue).

Razor Pages can be used alongside the MVC Framework, which is how I tend to use them. I write the main parts of the application using the MVC Framework and use Razor Pages for the secondary features, such as administration and reporting tools. You can see this approach in chapters 7–11, where I develop a realistic ASP.NET Core application called SportsStore.

1.1.3 Understanding Blazor

The rise of JavaScript client-side frameworks can be a barrier for C# developers, who must learn a different—and somewhat idiosyncratic—programming language. I have come to love JavaScript, which is as fluid and expressive as C#. But it takes time and commitment to become proficient in a new programming language, especially one that has fundamental differences from C#.

Blazor attempts to bridge this gap by allowing C# to be used to write client-side applications. There are two versions of Blazor: Blazor Server and Blazor WebAssembly. Blazor Server relies on a persistent HTTP connection to the ASP.NET Core server, where the application's C# code is executed. Blazor WebAssembly goes one step further and executes the application's C# code in the browser. Neither version of Blazor is suited for

all situations, as I explain in chapter 33, but they both give a sense of direction for the future of ASP.NET Core development.

1.1.4 Understanding the utility frameworks

Two frameworks are closely associated with ASP.NET Core but are not used directly to generate HTML content or data. Entity Framework Core is Microsoft's object-relational mapping (ORM) framework, which represents data stored in a relational database as .NET objects. Entity Framework Core can be used in any .NET application, and it is commonly used to access databases in ASP.NET Core applications.

ASP.NET Core Identity is Microsoft's authentication and authorization framework, and it is used to validate user credentials in ASP.NET Core applications and restrict access to application features.

I describe only the basic features of both frameworks in this book, focusing on the capabilities required by most ASP.NET Core applications. But these are both complex frameworks that are too large to describe in detail in what is already a large book about ASP.NET Core.

> **Topics for future editions**
>
> I don't have space in this book to cover every ASP.NET Core, Entity Framework Core, and ASP.NET Core Identity feature, so I have focused on those aspects that most projects require. If there are topics you think I should include in the next edition or in new deep-dive books, then please send me your suggestions at adam@adam-freeman.com.

1.1.5 Understanding the ASP.NET Core platform

The ASP.NET Core platform contains the low-level features required to receive and process HTTP requests and create responses. There is an integrated HTTP server, a system of middleware components to handle requests, and core features that the application frameworks depend on, such as URL routing and the Razor view engine.

Most of your development time will be spent with the application frameworks, but effective ASP.NET Core use requires an understanding of the powerful capabilities that the platform provides, without which the higher-level frameworks could not function. I demonstrate how the ASP.NET Core platform works in detail in part 2 of this book and explain how the features it provides underpin every aspect of ASP.NET Core development.

I have not described two notable platform features in this book: SignalR and gRPC. SignalR is used to create low-latency communication channels between applications. It provides the foundation for the Blazor Server framework that I describe in part 4 of this book, but SignalR is rarely used directly, and there are better alternatives for those few projects that need low-latency messaging, such as Azure Event Grid or Azure Service Bus.

gRPC is an emerging standard for cross-platform remote procedure calls (RPCs) over HTTP that was originally created by Google (the *g* in gRPC) and offers efficiency and scalability benefits. gRPC may be the future standard for web services, but it cannot be used in web applications because it requires low-level control of the HTTP messages that it sends, which browsers do not allow. (There is a browser library that allows gRPC to be used via a proxy server, but that undermines the benefits of using gRPC.) Until gRPC can be used in the browser, its inclusion in ASP.NET Core is of interest only for projects that use it for communication between back-end servers, such as in microservices development. I may cover gRPC in future editions of this book but not until it can be used in the browser.

1.2 Understanding this book

To get the most from this book, you should be familiar with the basics of web development, understand how HTML and CSS work, and have a working knowledge of C#. Don't worry if you haven't done any client-side development, such as JavaScript. The emphasis in this book is on C# and ASP.NET Core, and you will be able to pick up everything you need to know as you progress through the chapters. In chapter 5, I summarize the most important C# features for ASP.NET Core development.

1.2.1 What software do I need to follow the examples?

You need a code editor (either Visual Studio or Visual Studio Code), the .NET Core Software Development Kit, and SQL Server LocalDB. All are available for use from Microsoft without charge, and chapter 2 contains instructions for installing everything you need.

1.2.2 What platform do I need to follow the examples?

This book is written for Windows. I used Windows 10 Pro, but any version of Windows supported by Visual Studio, Visual Studio Code, and .NET Core should work. ASP.NET Core is supported on other platforms, but the examples in this book rely on the SQL Server LocalDB feature, which is specific to Windows. You can contact me at adam @adam-freeman.com if you are trying to use another platform, and I will give you some general pointers for adapting the examples, albeit with the caveat that I won't be able to provide detailed help if you get stuck.

1.2.3 What if I have problems following the examples?

The first thing to do is to go back to the start of the chapter and begin again. Most problems are caused by missing a step or not fully following a listing. Pay close attention to the emphasis in code listings, which highlights the changes that are required.

Next, check the errata/corrections list, which is included in the book's GitHub repository. Technical books are complex, and mistakes are inevitable, despite my best efforts and those of my editors. Check the errata list for the list of known errors and instructions to resolve them.

If you still have problems, then download the project for the chapter you are reading from the book's GitHub repository, https://github.com/manningbooks/pro-asp.net-core-7, and compare it to your project. I create the code for the GitHub repository by working through each chapter, so you should have the same files with the same contents in your project.

If you still can't get the examples working, then you can contact me at adam@adam-freeman.com for help. Please make it clear in your email which book you are reading and which chapter/example is causing the problem. Please remember that I get a lot of emails and that I may not respond immediately.

1.2.4 What if I find an error in the book?

You can report errors to me by email at adam@adam-freeman.com, although I ask that you first check the errata/corrections list for this book, which you can find in the book's GitHub repository at https://github.com/manningbooks/pro-asp.net-core-7, in case it has already been reported.

I add errors that are likely to cause confusion to readers, especially problems with example code, to the errata/corrections file on the GitHub repository, with a grateful acknowledgment to the first reader who reported them. I also publish a typos list, which contains less serious issues, which usually means errors in the text surrounding examples that are unlikely to prevent a reader from following or understanding the examples.

Errata bounty

Manning has agreed to give a free ebook to readers who are the first to report errors that make it onto the GitHub errata list for this book, which is for serious issues that will disrupt a reader's progress. Readers can select any Manning ebook, not just my books.

This is an entirely discretionary and experimental program. Discretionary means that only I decide which errors are listed in the errata and which reader is the first to make a report. Experimental means Manning may decide not to give away any more books at any time for any reason. There are no appeals, and this is not a promise or a contract or any kind of formal offer or competition. Or, put another way, this is a nice and informal way to say thank you and to encourage readers to report mistakes that I have missed when writing this book.

1.2.5 What does this book cover?

I have tried to cover the features that will be required by most ASP.NET Core projects. This book is split into four parts, each of which covers a set of related topics.

Part 1: Introducing ASP.NET Core

This part of the book introduces ASP.NET Core. In addition to setting up your development environment and creating your first application, you'll learn about the most important C# features for ASP.NET Core development and how to use the ASP.NET

Core development tools. Most of part 1 is given over to the development of a project called SportsStore, through which I show you a realistic development process from inception to deployment, touching on all the main features of ASP.NET Core and showing how they fit together—something that can be lost in the deep-dive chapters in the rest of the book.

PART 2: THE ASP.NET CORE PLATFORM

The chapters in this part of the book describe the key features of the ASP.NET Core platform. I explain how HTTP requests are processed, how to create and use middleware components, how to create routes, how to define and consume services, and how to work with Entity Framework Core. These chapters explain the foundations of ASP.NET Core, and understanding them is essential for effective ASP.NET Core development.

PART 3: ASP.NET CORE APPLICATIONS

The chapters in this part of the book explain how to create different types of applications, including RESTful web services and HTML applications using controllers and Razor Pages. These chapters also describe the features that make it easy to generate HTML, including the views, view components, and tag helpers.

PART 4: ADVANCED ASP.NET CORE FEATURES

The final part of the book explains how to create applications using Blazor Server, how to use the experimental Blazor WebAssembly, and how to authenticate users and authorize access using ASP.NET Core Identity.

1.2.6 What doesn't this book cover?

This book doesn't cover basic web development topics, such as HTML and CSS, and doesn't teach basic C# (although chapter 5 does describe C# features useful for ASP.NET Core development that may not be familiar to developers using older versions of .NET).

As much as I like to dive into the details in my books, not every ASP.NET Core feature is useful in mainstream development, and I have to keep my books to a printable size. When I decide to omit a feature, it is because I don't think it is important or because the same outcome can be achieved using a technique that I do cover.

As noted earlier, I have not described the ASP.NET Core support for SignalR and gRPC, and I note other features in later chapters that I don't describe, either because they are not broadly applicable or because there are better alternatives available. In each case, I explain why I have omitted a description and provide a reference to the Microsoft documentation for that topic.

1.2.7 How do I contact the author?

You can email me at adam@adam-freeman.com. It has been a few years since I first published an email address in my books. I wasn't entirely sure that it was a good idea, but I am glad that I did it. I have received emails from around the world, from readers

working or studying in every industry, and—for the most part anyway—the emails are positive, polite, and a pleasure to receive.

I try to reply promptly, but I get a lot of email, and sometimes I get a backlog, especially when I have my head down trying to finish writing a book. I always try to help readers who are stuck with an example in the book, although I ask that you follow the steps described earlier in this chapter before contacting me.

While I welcome reader emails, there are some common questions for which the answers will always be no. I am afraid that I won't write the code for your new startup, help you with your college assignment, get involved in your development team's design dispute, or teach you how to program.

1.2.8 What if I really enjoyed this book?

Please email me at adam@adam-freeman.com and let me know. It is always a delight to hear from a happy reader, and I appreciate the time it takes to send those emails. Writing these books can be difficult, and those emails provide essential motivation to persist at an activity that can sometimes feel impossible.

1.2.9 What if this book has made me angry and I want to complain?

You can still email me at adam@adam-freeman.com, and I will still try to help you. Bear in mind that I can only help if you explain what the problem is and what you would like me to do about it. You should understand that sometimes the only outcome is to accept I am not the writer for you and that we will have closure only when you return this book and select another. I'll give careful thought to whatever has upset you, but after 25 years of writing books, I have come to understand that not everyone enjoys reading the books I like to write.

Summary

- ASP.NET Core is a cross-platform framework for creating web applications.
- The ASP.NET Core platform is a powerful foundation on which application frameworks have been built.
- The MVC Framework was the original ASP.NET Core framework. It is powerful and flexible but takes time to prepare.
- The Razor Pages framework is a newer addition, which requires less initial preparation but can be more difficult to manage in complex projects.
- Blazor is a framework that allows client-side applications to be written in C#, rather than JavaScript. There are versions of Blazor that execute the C# code within the ASP.NET Core server and entirely within the browser.